ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths

Yeon-sup Lim
IBM T. J. Watson Research Center
y.lim@ibm.com

Erich M. Nahum
IBM T. J. Watson Research Center
nahum@us.ibm.com

Don Towsley
University of Massachusetts Amherst
towsley@cs.umass.edu

Richard J. Gibbens
University of Cambridge
richard.gibbens@cl.cam.ac.uk

ABSTRACT

Multi-Path TCP (MPTCP) is a new standardized transport protocol that enables devices to utilize multiple network interfaces. The default MPTCP path scheduler prioritizes paths with the smallest round trip time (RTT). In this work, we examine whether the default MPTCP path scheduler can provide applications the ideal aggregate bandwidth, i.e., the sum of available bandwidths of every paths. Our experimental results show that heterogeneous paths cause underutilization of the fast path, resulting in undesirable application behaviors such as lower streaming quality in a video than can be obtained using the available aggregate bandwidth. To solve this problem, we propose and implement a new MPTCP path scheduler, ECF (Earliest Completion First), that utilizes all relevant information about a path, not just RTT. We compare ECF with both the default and other MPTCP path schedulers, using both an experimental testbed and in-the-wild measurements. Our results show that ECF consistently utilizes all available paths more efficiently than other approaches under path heterogeneity, particularly for streaming video. In Web browsing workloads, ECF also does better in some scenarios and never does worse.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; *Experimentation*; • **Networks** → *Transport protocols*; *Packet scheduling*;

ACM Reference Format:
Yeon-sup Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. 2017. ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths. In *CoNEXT '17: The 13th International Conference on emerging Networking EXperiments and Technologies, December 12–15, 2017, Incheon, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3143361.3143376

1 INTRODUCTION

The recent advent of network devices with multiple wireless interfaces such as IEEE 802.11 (WiFi) and cellular (3G/LTE) is leading to efforts to take advantage of these interfaces and utilize multiple paths simultaneously. Multi-path TCP (MPTCP) is a new standardized transport protocol [7, 8] that takes advantage of such multiple network interfaces simultaneously and thus utilizes path diversity in the network.

One significant factor that affects MPTCP performance is the design of the path scheduler, which distributes traffic across available paths according to a particular scheduling policy. The default path scheduler of MPTCP is based on round trip time (RTT) estimates, that is, given two paths with available congestion window space, it prefers to send traffic over the path with the smallest RTT. While simple and intuitive, this scheduling policy does not carefully consider path heterogeneity, where available bandwidths and round trip times of the two paths differ considerably. This *path heterogeneity* is common in mobile devices with multiple interfaces [5, 10, 11, 18, 25] and can cause significant reorderings at the receiver-side [2, 5, 6, 16, 28]. To prevent this, MPTCP includes opportunistic retransmission and penalization mechanisms along with the default scheduler [22]. In long-lived flows, e.g., a single very large file transfer, MPTCP is able to enhance performance using these mechanisms. However, a large number of Internet applications such as Web browsing and video streaming usually generate traffic which consists of multiple uploads/downloads for relatively short durations. We find that in the presence of path heterogeneity, the default MPTCP scheduler is unable to efficiently utilize some paths with such a traffic pattern. In particular it does not take full advantage of the highest bandwidth paths, which should be prioritized to achieve the highest performance and lowest response time.

In this work, we propose a novel MPTCP path scheduler to maximize fast path utilization, called ECF (Earliest Completion First). To this end, ECF monitors not only subflow RTT estimates, but also the corresponding bandwidths (i.e., as embodied in the congestion windows) and the amount of data available to send (i.e., data queued in the send buffer). By determining whether using a slow path for the injected traffic will cause faster paths to become idle, ECF more efficiently utilizes the faster paths, maximizing throughput, minimizing download time, and reducing out-of-order packet delivery.

This paper makes the following contributions:

- We provide an analysis of the performance problems in MPTCP caused by path heterogeneity when using the default scheduler (§3). Using a streaming adaptive bit rate video workload, we illustrate how it does not utilize the aggregate available bandwidth and thus can lead to lower resolution video playback than is necessary.

CoNEXT '17, December 12–15, 2017, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5422-6/17/12…$15.00
https://doi.org/10.1145/3143361.3143376

- Based on this insight, we design a new path scheduler, Earliest Completion First (ECF), which takes path heterogeneity into account (§4). We provide an implementation of our scheduler in the Linux kernel.
- We evaluate ECF against the default MPTCP path scheduler and two other approaches, BLEST [6] and DAPS [16], in an experimental testbed (§5), across a range of bandwidths and round-trip times. We use multiple workloads: video streaming under fixed bandwidth (§5.2); video streaming under variable bandwidth (§5.3); simple file downloads (§5.4); and full Web page downloads (§5.5). We show how ECF improves performance by up to 30% above the other schedulers in heterogeneous path environments, improving fast path utilization and reducing out-of-order delivery, while obtaining the same performance in heterogeneous environments.
- To see how ECF works in real networks, we compare ECF against the default scheduler in the wild using the Internet (§6). We show improvements of 16% increased bit rates in video streaming (§6.2) and 26% reduction in completion times for full-page Web downloads (§6.3), while reducing out-of-order delay by up to 71%.

The rest of this paper is organized as follows. Section 2 provides the context for our work. We describe the problem of path underutilization with the default scheduler in Section 3. Section 4 presents the design of the ECF scheduler. Experimental results using the testbed are given in Section 5, while results measured over the Internet are provided in Section 6. Related work is reviewed in Section 7, and we conclude in Section 8.

2 BACKGROUND

2.1 Multi-path TCP

MPTCP splits a single data stream across multiple paths known as *subflows*, which are defined logically by all end-to-end interface pairs. For example, if each host has two interfaces, an MPTCP connection consists of four subflows. These subflows are exposed to the application layer as one standard TCP connection.

Since ordering is preserved within a subflow, but not across them, MPTCP must take care to combine subflows into the original ordered stream. MPTCP appends additional information called the data sequence number as a TCP header option to each packet. Based on the data sequence numbers, MPTCP merges multiple subflows properly and delivers in-order streams at the connection level.

When an MPTCP sender has data to send, it must choose a path over which to send that data. This is the task of the scheduler. The default MPTCP path scheduler selects the subflow with the smallest RTT for which there is available congestion window (CWND) space for packet transmission. In addition, to mitigate performance degradation with path heterogeneity, MPTCP includes opportunistic retransmission and penalization mechanisms, which can reinject unacknowledged packets from a slow subflow over a fast subflow and decreases CWND of the slow path.

2.2 Dynamic Adaptive Streaming over HTTP

Dynamic Adaptive Streaming over HTTP (DASH) [26] is the mechanism by which most video is delivered over the Internet. To stream videos with a bit rate appropriate for the available bandwidth, a DASH server provides multiple representations of a video content encoded at different bit rates. Each representation is fragmented into small video chunks that contain several seconds of video. Based on measured available bandwidth, a DASH client selects a chunk representation, i.e., bit rate, and requests it from a DASH server; this is called adaptive bit rate (ABR) selection.

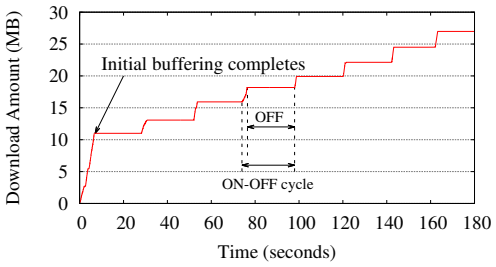


Figure 1: Example Download Behavior in Netflix

A DASH client player starts a streaming session with an initial buffering phase during which the player fills its playback buffer to some prescribed maximum level. During this phase, once the buffer reaches a second sufficient threshold, the player starts playing the video, and continues to retrieve video chunks until the initial buffering completes. After completing the initial buffering phase, the player pauses video download until the buffer level falls below the prescribed maximum level. If the playback buffer level falls below a prescribed minimum required to play out the video, the player stops playback and fills its buffer until it has a sufficient amount of video to begin playback again, which is called the rebuffering phase. This can lead to an ON-OFF traffic pattern where the player downloads chunks for a period of time and then waits until a specific number of chunks are consumed [23]. Figure 1 shows an example of client player download behavior when a mobile device fetches Netflix streaming video. This trace was collected using an Android mobile handset (Samsung Galaxy S3) while watching Netflix through WiFi on May 2014. During the OFF periods, the connection can go idle, causing CWND resets, as we will discuss in Section 3.

3 MOTIVATION

3.1 The Effect of Heterogeneous Paths

We first examine the effect of heterogeneous paths on application performance using adaptive video streaming, since it is currently one of the dominant applications in use over the Internet [24]. We measure the average video bit rate obtained by an Android DASH streaming client while limiting the bandwidth of the WiFi and LTE subflows on the server-side using the Linux traffic control utility tc [17] (full details of our experimental setup are given in Section 5.1). The streaming client uses a state-of-art adaptive bit rate selection (ABR) algorithm [12]. The choice of ABR does not significantly affect the results in this experiment as we use fixed bandwidths for each interface.

Table 1 presents the bit rates corresponding to each resolution. We choose bandwidth amounts slightly larger than those listed in Table 1, i.e., $\{0.3, 0.7, 1.1, 1.7, 4.2, 8.6\}$ Mbps, to ensure there is sufficient bandwidth for that video encoding.

Resolution	144p	240p	360p	480p	760p	1080p
Bit Rate (Mbps)	0.26	0.64	1.00	1.60	4.14	8.47

Table 1: Video Bit Rates vs. Resolution

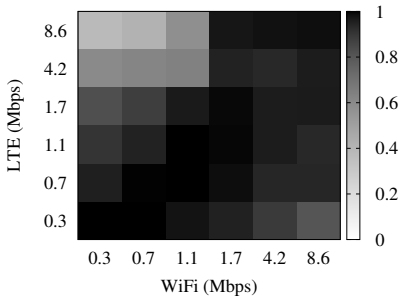


Figure 2: Ratio of Measured vs. Ideal Bit Rate Using MPTCP Default Path Scheduler (darker is better)

Figure 2 presents the ratio of the average bit rate achieved versus the ideal average bit rate available, based on the bandwidth combinations, when using the default MPTCP path scheduler. The figure is a grey-scale heat map where the darker the area is, the closer to the ideal bit rate the streaming client experiences. The closer the ratio is to one, the better the scheduler does in achieving the potential available bandwidth. The values are averaged over five runs. In a streaming workload, we define the ideal average bit rate as the minimum of the aggregate total bandwidth and the bandwidth required for the highest resolution at that bandwidth. For example, in the 8.6 Mbps WiFi and 8.6 Mbps LTE pair (the upper right corner in Figure 2), the ideal average bit rate is 8.47 Mbps, since the ideal aggregate bandwidth (8.6+8.6 = 17.2 Mbps) is larger than the required bandwidth for the highest resolution of 1080p (8.47 Mbps). Since the full bit rate is achieved, the value is one and the square is black.

Figure 2 shows that, when paths are significantly heterogeneous, the streaming client fails to obtain the ideal bit rate. For example, when WiFi and LTE provide 0.3 Mbps and 8.6 Mbps, respectively (the upper left box in Figure 2), the streaming client retrieves 480p video chunks, which requires only 2 Mbps, even though the ideal aggregate bandwidth is larger than 8.47 Mbps. Thus, the value is only 25% of the ideal bandwidth and the square is light grey. This problem becomes even more severe when the primary path (WiFi) becomes slower (compare the 0.3 Mbps & [0.3 – 8.6] Mbps and 8.6 Mbps & [0.3 – 8.6Mbps] pairs), as shown by the grey areas in the upper left and lower right corners.

Note that we observe similar performance degradation regardless of the congestion controller used (e.g., Olia [15]). In addition, the opportunistic retransmission and penalization mechanisms are enabled by default. This result shows that even with these mechanisms, the MPTCP default path scheduler does not sufficiently utilize the faster subflow when paths are heterogeneous.

3.2 Why Does Performance Degrade?

In this section, we identify the cause of the performance degradation when paths are heterogeneous. We investigate the TCP send buffer behavior of the faster subflow in the traces of the streaming

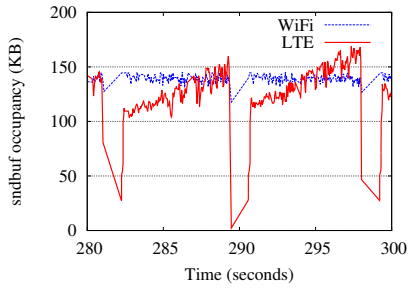


Figure 3: Send Buffer Occupancy (0.3 Mbps WiFi and 8.6 Mbps LTE. Including in-flight packets)

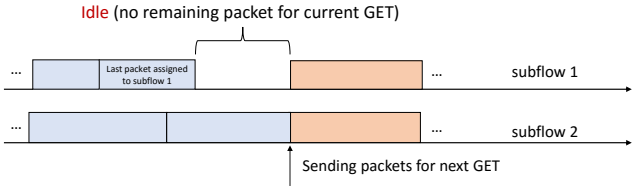


Figure 4: Case When Fast Subflow Becomes Idle

experiments. Figure 3 shows the send buffer occupancy (measured in the kernel) of the WiFi and LTE subflows when bandwidths are 0.3 and 8.6 Mbps, respectively. As can be seen, the streaming sender application periodically pauses to queue data into the LTE subflow, which has significantly higher bandwidth and lower RTT than the 0.3 Mbps WiFi subflow, and the LTE send buffer quickly empties due to acknowledgements. The streaming sender also pauses to use the WiFi subflow, i.e., the sender has no packet to send, but the sender is still transferring data over the slow WiFi subflow while the fast LTE subflow is idle. This shows that the application does not have any packet to send at that moment; the 8.6 Mbps LTE subflow completes its assigned packet transmissions much earlier than the 0.3 Mbps WiFi subflow and stays idle until the next download request is received.

Figure 4 presents a timing diagram to show how a fast subflow becomes idle, waiting until a slow subflow completes its assigned packet transmissions (here, subflow 1 is faster than subflow 2). To validate whether such an idle period really happens, we investigate the CDF of the time difference between the last packets over WiFi and LTE for four regulated bandwidth pairs. As shown in Figure 5, as paths become more heterogeneous, the time differences increase. In particular, the pause period (around 1 sec) in Figure 3 appears as the time difference of last packets. Note that this problem is due to the lack of packets to send, and not because of head of line blocking or receive window limitation problems discussed in [22].

Simple scheduling policies based solely on RTTs, e.g., allocating traffic to each subflow inversely proportional to RTT [16], cannot prevent this problem [13]. For example, consider two subflows where the RTTs are 10 ms and 100 ms, respectively, and the CWNDs of both subflows are 10 packets. Suppose the sender has 11 packets remaining to transmit; the initial CWND of Linux is 10 [3]. If a scheduler splits these 11 packets based on RTT, the fast subflow will

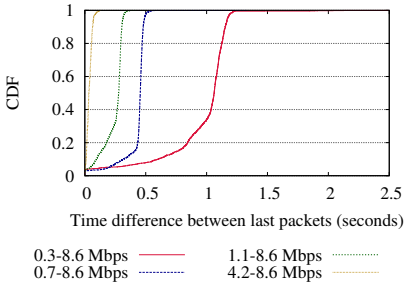


Figure 5: Time Difference of Last Packets

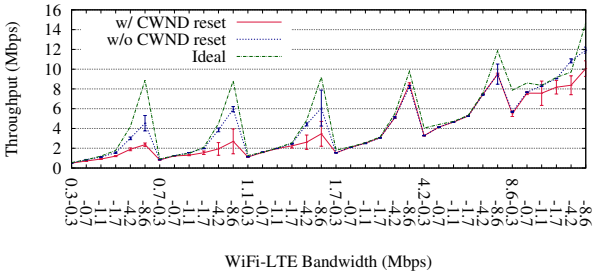


Figure 6: Throughput Measured at Streaming Client using Default Scheduler with/without CWND reset

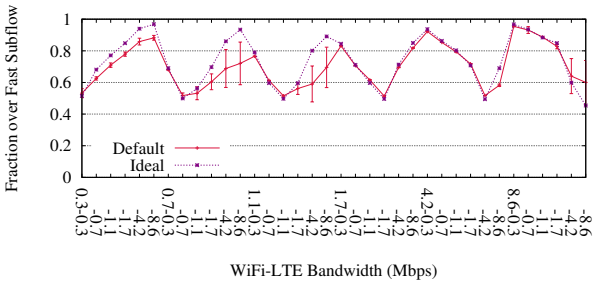


Figure 7: Fraction of Traffic Allocated to Fast Subflow using Default Scheduler in Streaming

complete 10 packet transmissions in one RTT (10 ms) and the slow subflow one packet in 100 ms. This results in a completion time of 100 ms, where the faster subflow is idle for 90 ms. In contrast, waiting for the 10 ms subflow to become available would result in completion time of just 20 ms. This shows that we must not only consider RTT, but also bandwidth and outstanding data on the subflow.

The performance degradation of these idle periods becomes more severe as an MPTCP connection is used for multiple object downloads. This is because the congestion controller resets the CWND to the initial window value and restarts from the slow-start phase if a connection is idle for longer than the retransmission timeout [1]. Since MPTCP congestion controllers such as coupled [27] and Olia [15] are designed to adapt a subflow CWND as a function of all the CWNDs across all subflows, resetting the CWND of a fast subflow because of an idle period can result in the fast subflow not being fully utilized for consecutive downloads.

To investigate the effect of the CWND reset, we perform streaming experiments with the default scheduler disabling the CWND reset. Figure 6 presents the average measured throughput according to bandwidth regulation pairs with and without the CWND reset. In this Figure, we also plot the aggregated bandwidths as the ideal throughputs. As shown in Figure 6, the default scheduler without the CWND reset achieves higher throughput that with the CWND reset, but the obtained throughput is still smaller than the ideal aggregate throughput. One of possible solutions to mitigate the performance degradation due to path heterogeneity can be just disabling the CWND reset. However, the problem is that since the CWND reset is to ensure correct detection of the amount of congestion in the network [1], we cannot disable this mechanism in congested network environments. In Section 4, we propose a scheduler to improve throughput in the presence of path heterogeneity without disabling the CWND reset.

Figure 7 presents the average fraction of traffic allocated to the fast subflow during the streaming experiments and the ideal fraction given the bandwidth pairs and corresponding measured average RTTs. As can be observed, the default scheduler places a smaller fraction of the traffic onto the fast subflow than the ideal model suggests. Together with the idle period of the fast subflow, this causes the aggregate throughput to degrade, resulting in a lower streaming quality selection than is possible given the available bandwidth.

4 APPROACH

To solve the performance degradation problem with path heterogeneity, we propose a new MPTCP path scheduler, called ECF (Earliest Completion First). ECF utilizes RTT estimates, path bandwidths (in the form of congestion window sizes), and the size of the send buffer at the connection-level.

An MPTCP sender stores packets both in its connection-level send buffer and in the subflow level send buffer (if the packet is assigned to that subflow). This means that if the number of packets in the connection level send buffer is larger than the aggregate number of packets in the subflow level send buffers, there are packets in the send buffer that need to be scheduled to the subflows.

Assume that there are k packets in the connection level send buffer, which have not been assigned (scheduled) to any subflow. If the fastest subflow in terms of RTT has available CWND, the packet can simply be scheduled to that subflow. If the fastest subflow does not have available space, the packet needs to be scheduled to the second fastest subflow.

We denote the fastest and the second fastest subflows as x_f and x_s, respectively. Let RTT_f, RTT_s and $CWND_f$, $CWND_s$ be the RTTs and CWNDs of x_f and x_s, respectively. If the sender waits until x_f becomes available and then transfers k packets through x_f, it will take approximately $RTT_f + \frac{k}{CWND_f} \times RTT_f$, i.e., the waiting and transmission time of k packets. Otherwise, if the sender sends some packets over x_s, the transmission will finish after RTT_s with or without completing k packet transfers. Thus, as shown in Figure 8, in the case of $RTT_f + \frac{k}{CWND_f} \times RTT_f < RTT_s$, using x_f after it becomes available can complete the transmission earlier than using

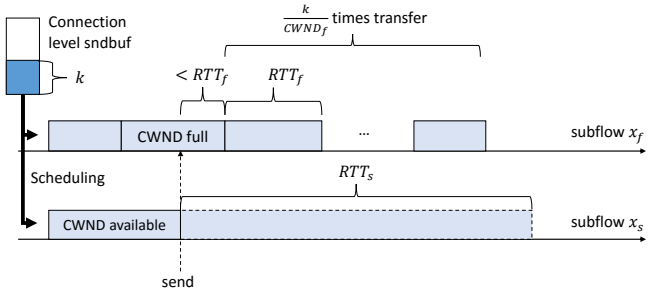


Figure 8: The case for waiting for the fast subflow

x_s at that moment. If $RTT_f + \frac{k}{CWND_f} \times RTT_f \geq RTT_s$, there are sufficient number of packets to send, so that using x_s at that moment can decrease the transmission time by utilizing more bandwidth than just by using x_f.

Based on this idea, we devise the ECF (Earliest Completion First) scheduler. Algorithm 1 presents the pseudo code for ECF. Note that the inequality uses RTT estimates and CWND values, which can vary over time. To compensate for this variability, we add a margin $\delta = \max(\sigma_f, \sigma_s)$, where σ_f and σ_s are the standard deviations of RTT_f and RTT_s, respectively, in the inequality for the scheduling decision:

$$\left(1 + \frac{k}{CWND_f}\right) \times RTT_f < RTT_s + \delta$$

This inequality takes into account the case in Figure 8, in which waiting for the fastest subflow completes transfer earlier than using the second fastest subflow. To more strictly assure this case, ECF checks an additional inequality, which validates if using the second fastest subflow with its CWND (it takes $\frac{k}{CWND_s} \times RTT_s$ to finish transfer) does not complete earlier than waiting for the fastest subflow (at least $2RTT_f$ for transfer),

$$\frac{k}{CWND_s} \times RTT_s \geq 2RTT_f + \delta$$

Here, we also use δ to compensate for RTT and CWND variabilities.

Note that ECF assumes that the subflows are in the congestion avoidance phase, which can cause incorrect estimations of the expected number of transfers (e.g., $\frac{k}{CWND_f}$) during the slow-start phase. However, x_f may quickly enter the congestion avoidance phase compared to x_s in the presence of path heterogeneity and subflows might be in the slow-start phase at the beginning of transfers, i.e., there are enough remaining packets to be transfered to utilize even x_s, resulting in negligible effect of the slow-start phase.

If these inequalities are satisfied, ECF does not use the second fastest subflow x_s and instead waits for the fastest subflow x_f to become available. ECF uses a different inequality for switching back to using x_s after deciding to wait for x_f:

$$\left(1 + \frac{k}{CWND_f}\right) \times RTT_f < (1 + \beta)(RTT_s + \delta).$$

This adds some hysteresis to the system and prevents it from switching states (waiting for x_f or using x_s now) too frequently.

Algorithm 1 ECF Scheduler

```
// This function returns a subflow for packet transmission
Find fastest subflow x_f with smallest RTT
if x_f is available for packet transfer then
    return x_f
else
    Select x_s using MPTCP default scheduler
    n = 1 + k / CWND_f
    δ = max(σ_f, σ_s)
    if n × RTT_f < (1 + waiting × β)(RTT_s + δ) then
        if k / CWND_s × RTT_s ≥ 2RTT_f + δ then
            waiting = 1        // Wait for x_f
            return no available subflow
        else
            return x_s
        end if
    else
        waiting = 0
        return x_s
    end if
end if
```

The implementation details of the ECF scheduler in the Linux Kernel are available at our technical report (http://cs.umass.edu/~ylim/mptcp_ecf).

5 EVALUATION IN A CONTROLLED LAB

In this section, we evaluate the ECF scheduler in a controlled lab setting. This lets us evaluate performance across a wide range of workloads and network configurations.

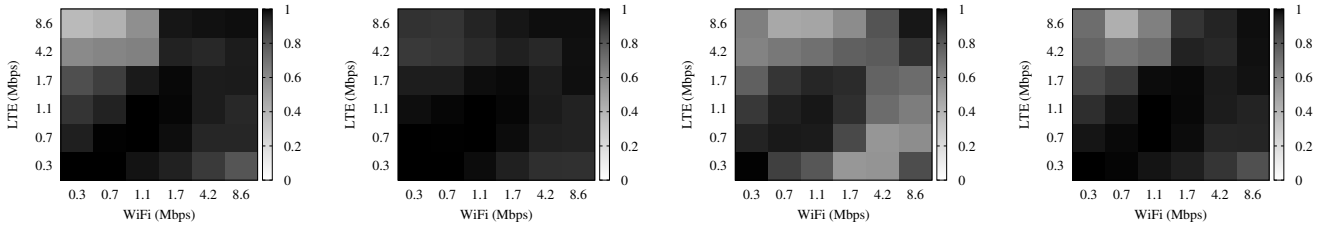
5.1 Experimental Setup

In our lab setting, we examine performance using three workloads: adaptive streaming video over HTTP, simple download activity using wget, and Web-browsing.

We use an Android mobile device (Google Nexus 5) as the client. Videos are played on the device using ExoPlayer [9]. The mobile device communicates with the server over the Internet using a WiFi access point (IEEE 802.11g) and an LTE cellular interface from AT&T. Note that MPTCP requires a default primary interface with which to initiate and receive transfers. While the choice of interface to use as the primary is a complex one [5], we use WiFi as the primary interface since that is the default in Android. The opportunistic retransmission and penalization mechanisms are enabled throughout all experiments.

For the server, we use a desktop running Ubuntu Linux 12.04 with the MPTCP 0.89 implementation deployed [19]. It is connected to the UMass campus network through a single Gigabit Ethernet interface. We use Apache 2.2.22 as the HTTP server while enabling HTTP persistent connections with the default Keep Alive Timeout (5 sec).

For DASH content, we select a video clip from [14] that is 1332 seconds long and encoded at 50 Mbps by an H.264/MPEG-4 AVC codec. The original resolution of the video is 2160p (3840 by 2160 pixels). We configure the streaming server to provide six representations of the video with resolutions varying from 144p to 1080p (just as Youtube does). We re-encode the video file at each resolution and



(a) Default (b) ECF (c) DAPS (d) BLEST

Figure 9: Ratio of Measured Average Bit Rate vs. Ideal Average Bit Rate (darker is better)

create DASH representations with 5 second chunks. Recall Table 1 in Section 3 presents the bit rates corresponding to each resolution.

The ECF hysteresis value β is set to 0.25 throughout our experiments (other values for β were examined but found to yield similar results, not shown due to space limitations). We compare ECF to the following schedulers:[1]

- Default: The default scheduler allocates traffic to a subflow with the smallest RTT and available CWND space. If the subflow with the smallest RTT does not have available CWND space, it chooses an available subflow with the second smallest RTT.
- Delay-Aware Packet Scheduler (DAPS) [16]: DAPS seeks in-order packet arrivals at the receiver by deciding the path over which to send each packet based on the forward delay and CWND of each subflow: DAPS assigns traffic to each subflow inversely proportional to RTT.
- Blocking Estimation-based Scheduler (BLEST) [6]: BLEST aims to avoid out-of-order delivery caused by sender-side blocking when there is insufficient space in the MPTCP connection-level send window. When this send window is mostly filled with packets over a slow subflow, the window does not have enough space, and the sender cannot queue packets to an MPTCP connection. To avoid this situation, BLEST waits for a fast subflow to become available, so that the fast subflow can transmit more packets during the slow subflow's RTT, so as to free up space of the connection-level send window.

Note that we do not examine MP-DASH [10], which has been proposed to schedule MPTCP path usage for video streaming since MP-DASH does not focus per-packet scheduling, exploiting information from a streaming client player; it activates and deactivates cellular paths according to required bandwidths to meet deadlines for chunk downloads regardless of path heterogeneity.

BLEST and ECF are similar in that both can decline opportunities to send on the slow subflow when it has available CWND space, but this decision is based on different design goals. BLEST's decision is based on the space in MPTCP send window and minimizing out-of-order delivery, whereas ECF's is based on the amount of data queued in the send buffer and with the goal to minimize completion time. We will show in Section 5.2.3 that ECF better preserves the faster flow's CWND and thus performs better.

[1]For DAPS and BLEST, we use the implementation from https://bitbucket.org/blest_mptcp/nicta_mptcp [6]

Bandwidth (Mbps)	0.3	0.7	1.1	1.7	4.2	8.6
WiFi RTT(ms)	969	413	273	196	87	40
LTE RTT(ms)	858	416	268	210	131	105

Table 2: Avg. RTT with Bandwidth Regulation

5.2 Video Streaming with Fixed Bandwidth

We begin by investigating whether ECF improves the performance of streaming applications compared to the other schedulers, while keeping bandwidth fixed for the duration of the experiment.

5.2.1 **Measured Bit Rate**. We first compare the schedulers based on achieved bit rate using our streaming workload. Figure 9 presents the ratio of the average bit rate of the default, ECF, DAPS and BLEST schedulers, normalized by the ideal average bit rate. Each experiment consists of five runs, where a run consists of the playout of the 20 minute video. The entries in Figure 9 are based on the average taken over the five runs. Table 2 shows the average RTT of each interface measured at sender-side based on the bandwidth configurations. Note that with the same bandwidth regulation, WiFi yields smaller RTTs than LTE, since the WiFi network is located in the UMass campus network and incurs lower delays than the AT&T LTE cellular network.

Figure 9(b) shows that ECF successfully enables the streaming client to obtain average bit rates closest to the ideal average bit rate, and does substantially better than the default when paths are not symmetric. Comparing Figure 9(c) with Figure 9(a), DAPS does not improve streaming performance; it yields even worse streaming bit rate than the default scheduler with some bandwidth configurations, e.g., 4.2Mbps for both of WiFi and LTE. Comparing Figure 9(d) with Figure 9(a), BLEST slightly improves streaming performance with 1 Mbps WiFi and [1..10] Mbps LTE pairs, but does not improve the average bit rate for other configurations.

5.2.2 **Traffic Split**. To understand why ECF performs better, we examine how each scheduler splits traffic to the fast subflow (i.e., the subflow providing higher bandwidth). Figure 10 shows the average fraction of traffic scheduled over the fast subflow for ECF, and BLEST schedulers (for clarity, DAPS is not included, as it performs the worst. Default is shown in Figure 7). As shown in Figure 10, ECF allocates traffic to the fast subflow close to the ideal allocation, compared to the other schedulers. By doing this, ECF obtains larger throughputs than other schedulers whenever path heterogeneity exists. This results in average bit rates close to the

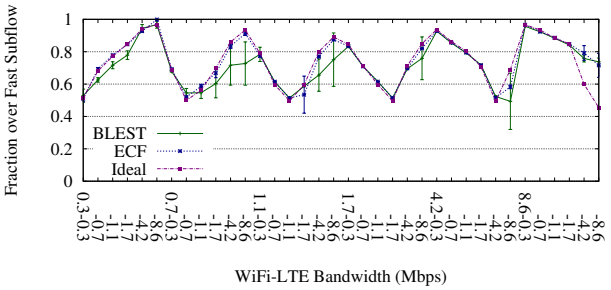


Figure 10: Fraction of Traffic Allocated to Fast Subflow in Streaming Workload - Fixed Bandwidth

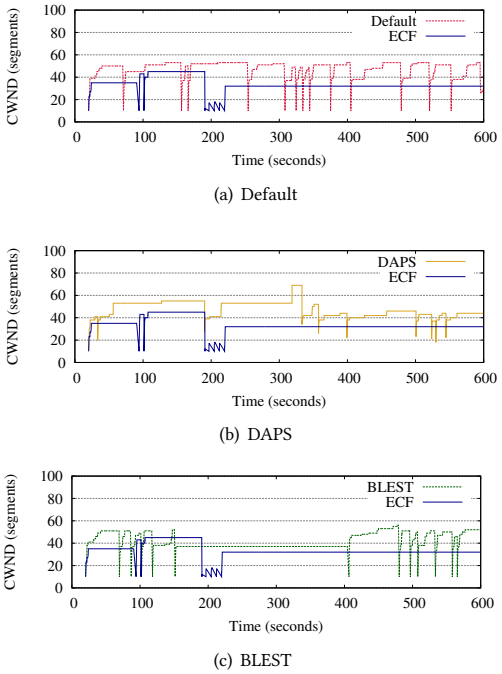
ideal average bit rate, as shown in Figure 9(b). Note that the fraction of traffic allocated to the fast subflow in the 8.6 Mbps WiFi and 8.6 Mbps LTE pair is larger than the ideal. This is because the 8.6 Mbps WiFi has a smaller RTT (40 ms) than the 8.6 Mbps LTE (105 ms) and transfer sizes (chunk downloads) are not large enough to fully utilize both subflows when bandwidths are large.

5.2.3 **Congestion Window Behavior**. Continuing our investigation, we study the behavior of the congestion window under the different schedulers. Figures 11 and 12 compare WiFi and LTE CWND behavior of the ECF and other schedulers when WiFi is 0.3 Mbps and LTE is 8.6 Mbps, a case where notable improvements by DAPS, BLEST, and ECF can be seen in Figure 9. As shown in Figures 11(a) and 12(a), the default scheduler (shown in dashed red curves) more aggressively utilizes the slower, smaller-bandwidth WiFi subflow, rather than the faster, larger-bandwidth LTE subflow. In contrast, ECF (solid blue curves) uses the LTE subflow more aggressively. This both makes more use of the faster subflow and reduces the number of idle periods, thus reducing the number of CWND resets due to idle periods, preserving the feasible values of the LTE CWND. Similarly, it makes less use of the WiFi subflow, as indicated in Figure 11 (a).

Similar CWND behaviors are seen with DAPS and BLEST, as shown in Figures 11(b)-(c) and 12(b)-(c). In Figures 11(b) and 12(b), while DAPS (shown in dashed yellow) outperforms the default scheduler, it does not exploit the faster subflow (LTE) as well as ECF. Figures 11(c) and 12(c) also show that ECF utilizes LTE more than BLEST (shown in dashed green), which operates better than default and DAPS. ECF yields the highest utilization of the LTE subflow, followed by BLEST, DAPS, and the default.

Note that the RTT of the LTE subflow (105 ms) is smaller than that of the WiFi subflow (969 ms) and that the idle period is more likely to happen at the LTE subflow with this bandwidth configuration. Thus, while the WiFi subflow frequently uses a maintained CWND, the LTE subflow unnecessarily starts with an initial CWND of 10 after the idle period in Figure 12. This results in underutilization of the fast LTE subflow due to the coupled operation of MPTCP congestion controller. Even with the smaller RTT of the LTE subflow, the BLEST, DAPS, and default schedulers cannot quickly increase the CWND in Figure 12.

To further study the behavior of the different schedulers in terms of the congestion window, we measure how often the CWND of the LTE subflows is reset to the initial window (IW) value, i.e., set back into slow start. Table 3 compares the average number of IW resets

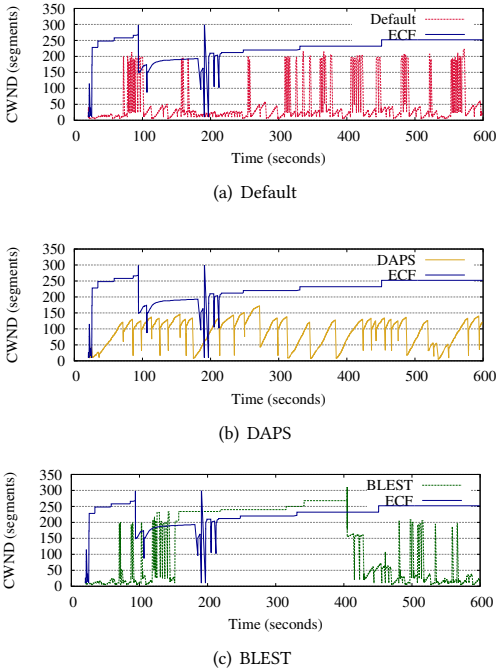


(a) Default

(b) DAPS

(c) BLEST

Figure 11: WiFi CWND Trace Comparison between ECF and Other Schedulers - 0.3 Mbps WiFi and 8.6 Mbps LTE



(a) Default

(b) DAPS

(c) BLEST

Figure 12: LTE CWND Trace Comparison between ECF and Other Schedulers- 0.3 Mbps WiFi and 8.6 Mbps LTE

Scheduler	Default	DAPS	BLEST	ECF
Average # of Events	486	92	382	16

Table 3: # of IW Resets - 0.3 Mbps WiFi & 8.6 Mbps LTE

over the entire video playback. Note that these resets are caused not only by idle timeouts, but by packet losses as well. As shown in Table 3, the default, DAPS, and BLEST schedulers experience high numbers of IW resets, while ECF incurs such events only 16 times on average.

5.2.4 **Out-of-Order Delay**. In the presence of path heterogeneity, MPTCP often causes out-of-order delays at the receiver-side, delaying delivery of arrived packets to the application layer. Since many Internet applications are sensitive to network quality metrics affected by out-of-order delays, most notably real-time streaming, it is important for MPTCP path schedulers to minimize out-of-order delays.

Figure 13 presents the CCDF of the out-of-order delay that individual packets experience with the default scheduler. As shown in Figure 13, the default scheduler yields larger out-of-order delays as paths become more heterogeneous. The median delay is a full second in the case of 0.3 Mbps WiFi and 8 .6 Mbps LTE. In addition, we observe that out-of-order delay is strongly related to the time difference between the last packets (compare Figures 14 and 3). In other words, the larger time difference of last packets is likely to be triggered by larger out-of-order delay at the end of download completion.

Figure 14 compares the CCDF of out-of-order delay of each scheduler under two bandwidth configurations: a heterogeneous one with 0.3 Mbps WiFi and 8.6 Mbps LTE as shown in Figure

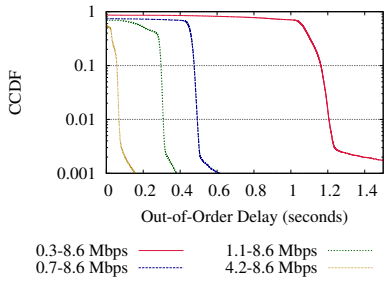
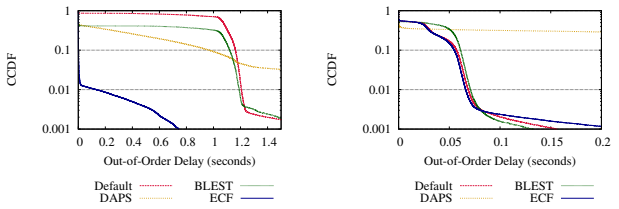


Figure 13: Out-of-Order Delay (Default Scheduler)



(a) 0.3 Mbps WiFi and 8.6 Mbps LTE (b) 4.2 Mbps WiFi and 8.6 Mbps LTE

Figure 14: Out-of-Order Delay - Streaming

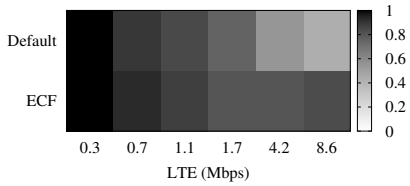


Figure 15: Ratio of Measured vs. Ideal Bit Rate of Default and ECF Scheduler when using 4 subflows (darker is better)

14(a), and a relatively symmetric one with 4.2 Mbps WiFi and 8.6 Mpbs LTE. as shown in Figure 14(b). Note in the heterogeneous configuration, DAPS, BLEST, and ECF all yield smaller out-of-order delays than the default scheduler, with ECF performing the best. Under ECF, almost 99.9% of packets experience out-of-order delays less than 0.8 seconds. In contrast, with the default scheduler, over 99% of the packets suffer from out-of-order delays larger than one second, while DAPS and BLEST have 90% and 96% of packets. In the symmetric configuration shown in Figure 14(b), out-of-order delay becomes much smaller (note the x-axis relative to 14(a)), with little difference between the schedulers (except for DAPS) as there is little path heterogeneity. The schedulers mostly yield out-of-order delays of less than 0.1 seconds (again, except for DAPS). DAPS, on the other hand, delivers over 60% of packets to the application layer with delays greater than 0.05 sec, which is worse than that even the default scheduler.

5.2.5 **More Subflows**. To validate whether ECF works for more than two subflows, we compare the performance of the default and ECF scheduler for bandwidth pairs of 0.3 Mbps and [0.3-8.6] Mbps using four subflows (two over WiFi and two over LTE), with the default scheduler using two subflows that experiences significant performance degradation with path heterogeneities. For these experiments, we regulate the subflows over each interface to evenly provide designated bandwidths, i.e., each WiFi subflow bandwidth is limited to 0.15 Mbps for 0.3 Mbps WiFi. Figure 15 presents the ratio of the average measured bit rate over the ideal bit rate. As shown in Figure 15, ECF mitigates performance degradation in the presence of significant path heterogeneity.

5.3 Video Streaming with Bandwidth Changes

The previous section studied video streaming performance when bandwidths are stable. In this section we examine how ECF and the other schedules respond to changes in network bandwidth. Here, we change WiFi and LTE bandwidths randomly at exponentially distributed intervals of time with an average of 40 seconds. The bandwidth values are selected from the set $\{0.3, 1.1, 1.7, 4.2, 8.6\}$ Mbps, and chosen uniformly at random. Ten scenarios are generated, each using a different unique random seed, with throughputs measured at the streaming client, averaged over 5 runs per scenario.

Figure 16 compares the average throughputs seen using the default, ECF, and BLEST schedulers for each random scenario (DAPS consistently performs worse than the default and is omitted for clarity in the Figure). Note the error bars of one standard deviation, indicating variability even when the same seed is used. As can be seen, ECF outperforms the other schedulers in terms of average

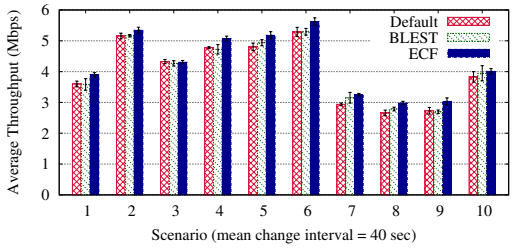


Figure 16: Streaming Throughputs - Random Bandwidth Changes

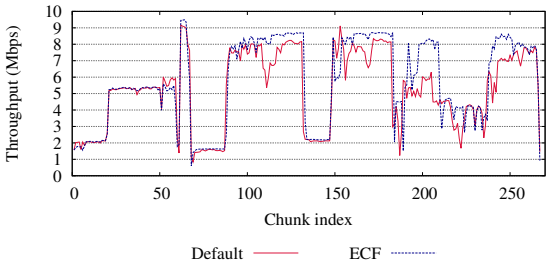


Figure 17: Example Throughput Trace - Streaming with Random Bandwidth Changes

throughput, producing the highest average streaming bit rate. Similar behavior is seen with the average bit rate (omitted due to space limitations).

Recall that ECF makes more efficient use of the faster subflow when path heterogeneity exists, and otherwise yields at least similar performance as the default scheduler; therefore the ECF gain in these experiments depends on how often path heterogeneity appears in a random bandwidth scenario. To see performance during bandwidth changes, Figure 17 presents measured throughput for each chunk download for a particular random scenario (scenario 6 in Figure 16). We observe that ECF yields similar or larger download throughputs than the default scheduler for any streaming chunk download. In particular, ECF obtains up to 2x more throughput than the default scheduler in the presence of path heterogeneity (e.g., 200th chunk download).

5.4 Simple Web Downloads

In this Section we examine the performance of the four schedulers for simple file downloads using wget. The purpose of these experiments is to show that ECF improves performance in the presence of path heterogeneity for this workload, without degrading performance when paths are heterogeneous. We measure the wget download completion time for several file sizes (64 KB to 2 MB, in powers of two) while regulating the WiFi and LTE bandwidths between [1,10] Mbps in a manner similar to Section 3.1. In these experiments, since MPTCP transfers a single object during a comparatively shorter time than the streaming experiments, we do not expect performance differences across the schedulers to be significant; an idle period of the fast subflow only appears once and a CWND reset after idle never occurs.

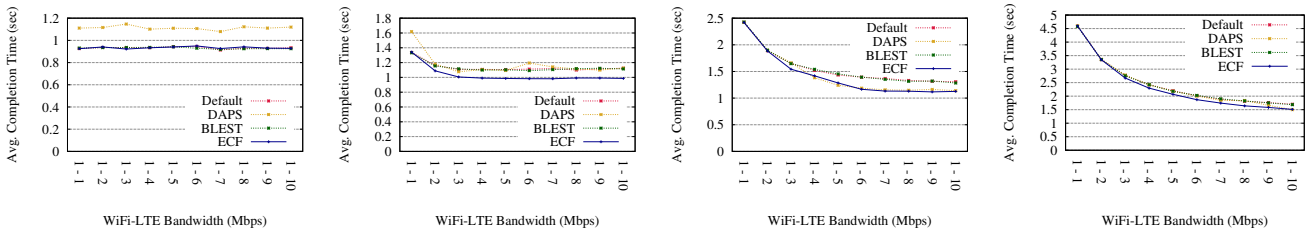
Figure 18 presents a set of download completion times for 128 KB, 256 KB, 512 KB, and 1 MB files for a range of bandwidth configurations, where WiFi is 1 Mbps and LTE varies from 1 to 10 Mbps, for all the schedulers, averaged over thirty runs. For configurations with WiFi greater than 1 Mbps, we observe no statistical differences between the schedulers (except for DAPS, which frequently performs worse), and thus omit those figures for space limitations. Recall that WiFi is the primary subflow. MPTCP rarely utilizes a secondary subflow (LTE in this case) for small transfers [2]. Therefore, unless the primary path (WiFi) is extremely slow, path schedulers do not affect performance for small downloads such as 128 KB. However, DAPS sometimes yields larger average completion times, e.g., the 128 KB case where WiFi is 1 Mbps and LTE ranges from [1, 10] Mbps. We attribute this to DAPS strong dependency on the RTT ratio; an incorrect estimate of the LTE RTT results in unnecessary trials to inject traffic into the slow LTE subflow. Figure 18 shows that ECF does no worse statistically than the default scheduler, and occasionally does better when paths are heterogeneous with larger transfers than 256 KB. For example, when LTE is 10 Mbps and WiFi is 1 Mbps, ECF reduces download time by 200 ms or 13%.

To compare performance between the default and ECF schedulers in more detail, Figure 19 shows the download completion time of the ECF scheduler normalized relative to that of the default scheduler. To plot this figure, we set the normalized value to one if the download time difference between the ECF and default scheduler is within the range of their standard deviations. Otherwise, the ratio is defined as the ratio of the averages. Thus, the value of one in Figure 19 means that both of the default and ECF schedulers yield similar performance (shown as white), and smaller than one means that the ECF scheduler takes less time than the default scheduler (shown as more blue). If ECF ever did worse than the default, that ratio would be expressed in degrees of red, but that does not happen.

As shown in Figure 19 (a), for small transfers (128 KB), the default and ECF schedulers both yield the same completion time. We observe notable performance differences between the ECF and default schedulers for downloads of 256 KB and larger. Figures 19(b)-(c) show that ECF yields up to 20% smaller download times than the default scheduler in the presence of path heterogeneity when downloading files of 256 KB or larger. Note that the relative improvement by ECF decreases as the transfer size increases. This is because, in these experiments, a single object is downloaded over an MPTCP connection. An idle period of the fast subflow only happens once, and the total transfer time becomes comparatively longer than this idle period with a larger transfer.

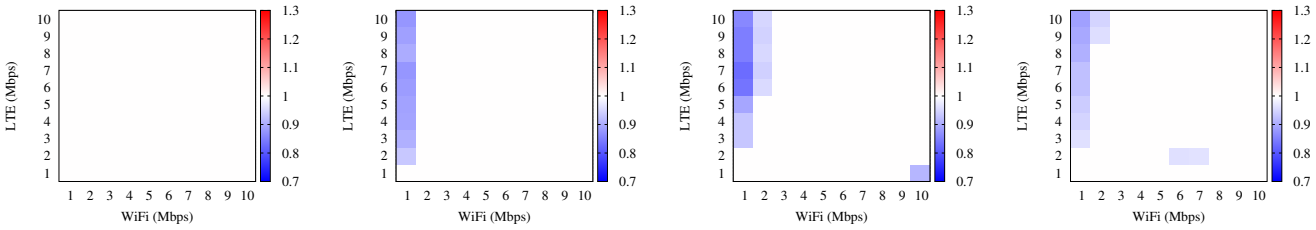
5.5 Web Browsing

We now examine ECF performance in our third workload, Web browsing. We deploy a copy of CNN's home page (as of 9/11/2014) consisting of 107 Web objects into our MPTCP server. Web-browsing is similar to a series of consecutive wget downloads, except that a persistent connection is used. Thus, consecutive downloads over one MPTCP connection are more susceptible to idle timeouts and the corresponding CWND resets, compared to a single object download using wget. To see how each scheduler affects Web object download performance, we examine the distribution of object download



(a) 128KB (b) 256KB (c) 512KB (d) 1M

Figure 18: Average Download Completion Time - 128 KB, 256 KB, 512 KB, and 1 MB (lower is better)



(a) 128KB (b) 256KB (c) 512KB (d) 1M

Figure 19: ECF Average Download Completion Time Ratio Normalized by Default

completion time while regulating the WiFi and LTE bandwidths between [1,10] Mbps as in Section 5.4. In this experiment, the Android Web browser establishes six parallel (MP)TCP connections to the server (12 subflows for MPTCP), using persistent HTTP connections. Note that we collect traces from 10 runs.

Figure 20 compares the CCDFs of individual object download completion times of each scheduler across three bandwidth configurations of varying heterogeneity. In Figure 20(a), with equal bandwidth (5.0 Mbps), we see that all schedulers yield almost the same download completion time: 98% of object downloads are completed in a similar time for all schedulers. In Figure 20(b), with 1.0 Mbps WiFi and 5.0 Mbps LTE, ECF completes 99% of object downloads earlier than the other schedulers. In this configuration, BLEST yields almost the same performance as the default scheduler and DAPS does not achieve any performance gain, as was the case in the streaming and simple Web download experiments. In Figure 20(c), with 1.0 Mbps WiFi and 10.0 Mbps LTE, we observe that as paths become more heterogeneous, ECF again explicitly exhibits smaller object download completion times than the other schedulers, while DAPS and BLEST do not outperform the default scheduler.

Figure 21 presents the CDFs of the out-of-order delay that individual packets experience while the browser downloads Web objects under the three bandwidth configurations. As with Figure 14 in the streaming cases, we observe that ECF successfully reduces out-of-order delay in Web browsing activities when paths are heterogeneous.

6 EVALUATION IN THE WILD

We next examine whether ECF provides better performance than the default scheduler in more realistic environments. This lets us see whether the conditions we identify in Section 5 actually occur in the wild. We limit our comparison of ECF to just the default scheduler since the other schedulers do not exhibit consistent improvement over the default scheduler in the previous experiments.

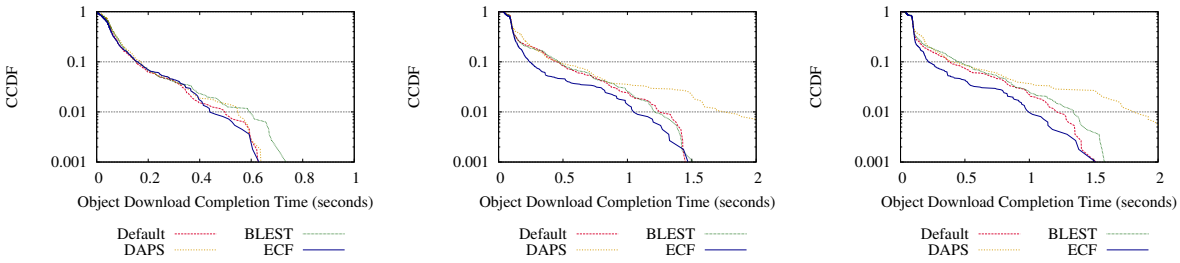
6.1 Experimental Setup

In this experiment, we deploy an MPTCP enabled server in Washington D.C. using a commercial cloud provider, which uses the same server configuration for the controlled in-lab experiments described in Section 5.1. The mobile device communicates with the server over the Internet using a WiFi access point (a local town public WiFi) and an LTE cellular interface from AT&T. Note that in these experiments, the device uses each network as-is without any additional bandwidth regulation.

6.2 Video Streaming in the Wild

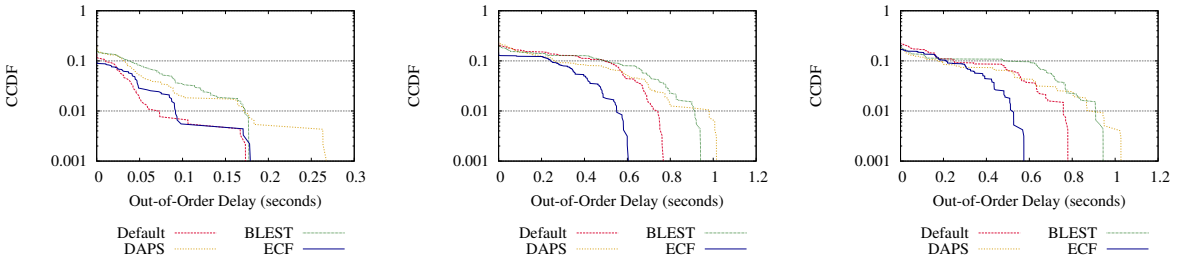
We first explore the streaming performance over MPTCP using the default and ECF schedulers in the wild configuration. We perform nine runs over two days using our streaming workload on the WDC server. Figure 22(a) shows the average measured RTT for each run. Note that RTT varies widely over the two days and results are sorted by WiFi average RTT. We observe that LTE has a consistent average RTT (around 70 ms). As shown in Figure 22(a), runs 1 and 2 have similar WiFi and LTE RTTs, that is, both paths are symmetric in terms of RTT, resulting in similar performance between the default and ECF schedulers. Since there are significant differences between WiFi and LTE RTTs in runs 4-9, the default scheduler is likely to experience throughput degradation whereas ECF is not.

Figure 22(b) presents the average throughputs obtained by the streaming client using the default and ECF schedulers. As expected,



(a) 5.0 Mbps WiFi and 5.0 Mbps LTE (b) 1.0 Mbps WiFi and 5.0 Mbps LTE (c) 1.0 Mbps WiFi and 10.0 Mbps LTE

Figure 20: Web Object Download Completion Time



(a) 5.0 Mbps WiFi and 5.0 Mbps LTE (b) 1.0 Mbps WiFi and 5.0 Mbps LTE (c) 1.0 Mbps WiFi and 10.0 Mbps LTE

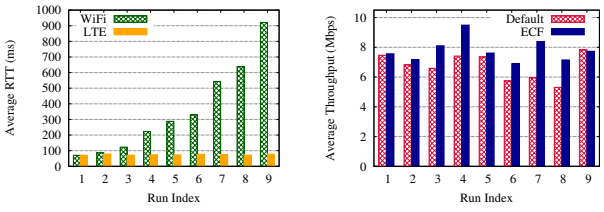
Figure 21: Comparison of Out-of-Order Delay - Web Browsing

both schedulers yield similar throughputs in runs 1 and 2. In later runs, the differences in RTT between WiFi and LTE become larger, resulting in larger average throughputs for ECF than for the default scheduler. However, both schedulers again obtain similar average throughputs in run 9. Note that the WiFi subflow is the primary subflow and in run 9, the WiFi average RTT is close to one second, which is more than ten times larger than the LTE RTT. We observe that the default scheduler injects 3% of packets through the WiFi subflow in this case: those packets might be the first few packets at the beginning of the HTTP GET responses and in this run, the WiFi subflow occasionally completes these transfers before the LTE subflow transmits the last packet, infrequently affecting the performance of LTE subflow (in terms of LTE throughput, the default scheduler yields 7.31 Mbps while ECF does 7.72 Mbps). On average, the ECF throughput is 7.79 Mbps while the default scheduler 6.72 Mbps, an improvement of 16%.

6.3 Web Browsing in the Wild

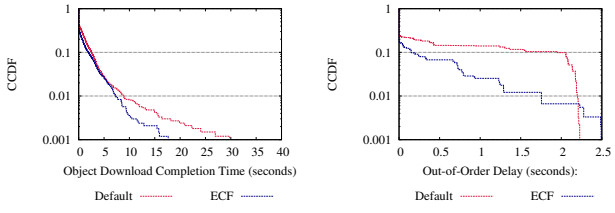
Next, we investigate the distribution of the object download completion times when the device retrieves a copy of CNN's home page at the WDC server, measured over thirty runs. Figure 23(a) compares the CCDFs of the individual Web object download completion times of the default and ECF schedulers. The average statistics are listed in Table 4. As shown in Figure 23(a), ECF yields smaller object download completion times than the default scheduler. On average, ECF completes the object downloads in 0.65 seconds, while the default scheduler requires 0.88 seconds, an improvement of 26%. In addition, while ECF completes 99.9% of object downloads in around 17 seconds, the default scheduler requires 30 seconds.

Figure 23(b) presents the CCDFs of the out-of-order delay that individual packets experience. As shown in Figure 23(b), 99% of packets downloaded using ECF experience smaller out-of-order



(a) RTT (b) Throughput

Figure 22: Streaming Experiments in the Wild



(a) Download Completion Time (b) Out-of-Order Delay

Figure 23: Web Browsing Comparison in the Wild

	Download Completion Time (sec)	Out of Order Delay (sec)
Default	0.882	0.297
ECF	0.650	0.087
ECF Improvement	26% shorter	71% shorter

Table 4: Average Statistics of Web Browsing in the Wild

delays than with the default scheduler. ECF yields an average out-of-order delay of 0.087 seconds, while the default scheduler yields an average of 0.297 seconds, an improvement of 71%. Only 0.2% of packets downloaded using ECF exhibit slightly larger out-of-order delays than the largest one using the default scheduler. We found that 0.2% is from twelve instances out of approximately 27000 data points; these twelve packets suffer out-of-order delays of approximately 2.5 seconds.

7 RELATED WORK

Although the design of the MPTCP path scheduler significantly impacts performance and quality of experience, there have not been many practical studies of an improved MPTCP path scheduler that have been implemented and evaluated experimentally.

Raiciu et al. [22] points out that path heterogeneity can result in performance degradation due to head of line blocking or limited receive window size due to reorderings. To resolve these problems, they propose opportunistic retransmission and penalization mechanisms, which are included in the Linux MPTCP Kernel implementation. These mechanisms have been evaluated in more detail in [20, 21].

Kuhn et al. [16] propose delay-aware packet scheduling for MPTCP. This approach considers large path heterogeneity in delay and stable CWND, but does not take advantage of information from the send buffer. In addition, it is evaluated only by ns2 simulations.

Ferlin et al. [6] propose a scheduler to prevent fast subflow blocking due to path heterogeneity. Their scheduler waits for a fast subflow if during the RTT of the slow path, the fast subflow can transfer more packets than the available space in the connection-level send window. However, it does not consider idle fast subflow due to nothing to send.

Yang et al. [28] propose a scheduler that distributes traffic proportional to the estimated path capacity. They only consider scenarios with very large transfers in a network with a small amount of buffering.

Yang et al. [29] suggest another scheduler similar to [28], which chooses paths that finishes transfers of scheduled packets in shortest time. However, they assume that packets can be injected to even unavailable paths (no space in CWND), which can cause unnecessary retransmissions if those paths are actually unavailable, e.g., disconnected.

Corbillion [4] proposes a scheduler to improve streaming video performance over MPTCP. They do not implement their approach and evaluate it only via simulation. In addition, their solution requires modifying the video sending application to integrate it with the MPTCP scheduler, whereas our approach is application-independent.

Nikravesh et al. [18] present a measurement study of MPTCP in the wild, and propose MPFLEX, an architecture for supporting multipath over mobile networks. However, MPFLEX is not compatible with MPTCP and requires modifications to both the client and server.

Han et al. [10] present MP-DASH, a framework for scheduling streaming video traffic over MPTCP. They show that by exploiting knowledge of video streaming, traffic can be scheduled so as to significantly reduce cellular usage and power consumption with negligible degradation of QoE. Their approach, however, requires modifications to both the client and server, and is focused solely on video traffic. ECF, in contrast, is a server-side only modification, improving deployability, and works transparently for multiple workloads, not just streaming video.

8 CONCLUSION

In this work, we show that the default MPTCP path scheduler degrades performance in the presence of path heterogeneity. We identify the root cause of the problem: faster paths are under-utilized due to idle periods and the consequent CWND resets. We propose a novel MPTCP path scheduler ECF to improve the utilization of the fastest path. We compare ECF with the default, DAPS, and BLEST MPTCP path schedulers, using both an experimental testbed and in-the-wild measurements. Our experimental results show that ECF outperforms the existing schedulers across a range of workloads when path heterogeneity is significantly large, while providing the same performance using homogeneous paths.

ACKNOWLEDGEMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] M. Allman, V. Paxson, and E. Blanton. TCP congestion window validation. RFC 5681, 2009.
[2] Y.-C. Chen, Y.-S. Lim, R. J. Gibbens, E. Nahum, R. Khalili, and D. Towsley. A measurement-based study of Multipath TCP performance in wireless networks. In *Proc. of ACM IMC*, pages 455–468, Nov 2013.
[3] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing tcp's initial window. RFC 6928, 2013.
[4] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. Cross-layer scheduler for video streaming over MPTCP. In *Proc. of ACM MMSys*, pages 7:1–7:12, 2016.
[5] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or both? Measuring multi-homed wireless Internet performance. In *Proc. of ACM IMC*, pages 181–194, 2014.
[6] S. Ferlin, O. Alay, O. Mehani, and R. Boreli. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *Proc. of IFIP Networking*, pages 1222–1227, 2016.
[7] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural guidelines for multipath TCP development. RFC 6182 (Informational), Mar. 2011.
[8] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, 2013.
[9] Google. Exoplayer. http://google.github.io/ExoPlayer/.

[10] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proc. of ACM CoNEXT*, pages 129–143, 2016.
[11] J. Huang, Q. Feng, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proc. of ACM MobiSys*, pages 225–238, 2012.
[12] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of ACM SIGCOMM*, pages 187–198, 2014.
[13] J. Hwang and J. Yoo. Packet scheduling for multipath tcp. In *International Conference on Ubiquitous and Future Networks*, pages 177–179, 2015.
[14] JackFrags. 4k gaming montage. http://4ksamples.com/4k-gaming-montage/.
[15] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec. MPTCP is not pareto-optimal: Performance issues and a possible solution. In *Proc. of ACM CoNEXT*, pages 1–12, 2012.
[16] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. DAPS: Intelligent delay-aware packet scheduling for multipath transport. In *Proc. of IEEE ICC*, pages 1222–1227, 2014.
[17] Linux Foundation. Linux advanced routing and traffic control. http://lartc.org/howto/.
[18] A. Nikravesh, Y. Goo, F. Qian, Z. M. Mao, and S. Sen. An in-depth understanding of multipath TCP on mobile devices: Measurement and system design. In *Proc. of ACM MobiCom*, pages 189–201, 2016.
[19] C. Paasch and S. Barre. Multipath TCP in the Linux kernel. http://www.multipath-tcp.org.
[20] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of multipath TCP schedulers. In *Proc. of ACM SIGCOMM Workshop on Capacity Sharing Workshop*, pages 27–32, 2014.
[21] C. Paasch, R. Khalili, and O. Bonaventure. On the benefits of applying experimental design to improve multipath TCP. In *Proc. of ACM CoNEXT*, pages 393–398, 2013.
[22] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? Designing and implementing a deployable multipath TCP. In *Proc. of USENIX NSDI*, pages 399–412, 2012.
[23] A. Rao, Y.-s. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. of ACM CoNEXT*, pages 25:1–25:12, 2011.
[24] Sandvine. Global Internet phenomena report, Latin America and North America 2016. https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf.
[25] J. Sommers and P. Barford. Cell vs. wifi: on the performance of metro area mobile connections. In *Proc. of ACM IMC*, pages 301–314. ACM, 2012.
[26] T. Stockhammer. Dynamic adaptive streaming over HTTP – Standards and design principles. In *Proc. of ACM MMSys*, pages 133–144, 2011.
[27] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. of USENIX NSDI*, pages 99–112, 2011.
[28] F. Yang, P. Amer, and N. Ekiz. A scheduler for multipath TCP. In *Proc. of ICCCN*, pages 1–7, 2013.
[29] F. Yang, Q. Wang, and P. D. Amer. Out-of-order transmission for in-order arrival scheduling for multipath tcp. In *International Conference on Advanced Information Networking and Applications Workshops*, pages 749–752, 2014.